Erschienen in: LREC-2008 Workshop on NLP Resources, Algorithms and Tools for Authoring Aids, Marrakech, Morocco, 1 Juni 2008 - 1 Juni 2008, 9-13.

Language-Aware Text Editing

Cerstin Mahlow, Michael Piotrowski, Michael Hess

Institute of Computational Linguistics, University of Zurich
Binzmühlestrasse 14, 8050 Zürich, Switzerland
{mahlow, mxp, hess}@cl.uzh.ch

Abstract

While software developers have various "power tools" at their disposal that make the writing of computer programs more efficient, authors of texts do not have the support of such tools. Text processors still operate on the level of characters and strings rather than on the level of word forms and grammatical constructions. This forces authors to constantly switch between low-level, character oriented, editing operations and high-level, conceptual, verbalisation processes. We suggest the development of language-aware text editing tools that simplify certain frequent, yet complex editing operations by defining them on the level of linguistic units. Pluralizing an entire noun phrase plus the verb forms governed by it would be an ambitious example, swapping the elements of a conjunctive construction a more modest one.

We propose a taxonomy for revising and editing operations with respect to revising and editing as such as well as engineering desiderata. We describe the components of a pilot implementation for German where these operations are seamlessly integrated with the standard functions of an existing open-source editor. The operations can be invoked on demand and do not intrude on the authoring process. Changes can be performed locally or globally, thus simplifying the writing process considerably, and making the resulting texts more consistent.

1. Introduction

The process of creating a high-quality text involves several iterations of writing, revising and editing. Ever since the beginnings of computerized word processing in the 1960s (see Haigh (2006) for a historical overview) researchers tried to develop tools to support writers. In the history of text processing aids there were several attempts (see, e.g., Oakman (1994)) to create expert tools for expert writers, but today's word processors mainly operate on the level of characters rather than linguistic units. The result are typical revision and editing errors such as duplicate verbs or extraneous conjunctions.

In this paper we propose functions for word processors that aim at improving the "brain-to-hand-to-keyboard-to-screen-connection" (Taylor, 1987, p. 79). We will show that editors can be upgraded to provide functions that operate on linguistic elements with relatively low costs in terms of linguistic resources. We will outline what aspects need to be considered when implementing these functions.

2. Motivation

We think that many of the revision and editing errors are caused by *attentional disruption* since writers have to translate their high-level goals (e.g., changing the grammatical mood of an expression) into the low-level, character-oriented functions of the editor.

Even simple revisions, such as changing "editing and revising" to "revising and editing", which *conceptually* merely consists of swapping the conjuncts, require substantial planning and memory capacity when they have to be executed in an editor or word processor. [1]

However, as McCutchen (1996) points out, the capacity of a writer's working memory and, of course, cognitive resources in general are limited, and when resources are diverted to other activities, this will have a negative impact on writing processes (i.e., planning, translating, and reviewing).

In this paper we will concentrate on texts in German. German is an interesting language for our research because, as a heavily inflectional language, it is morphologically much richer than English, and this has implications for revising and editing. For example, globally replacing one word with another, say, *hut* with *tent*, doesn't pose any problem in English, since there are only three word forms and no changes to the stems. Apart from a very small number of exceptions (such as *mouse* or *foot*), all replacements can thus be done by simple regular expressions. Inflectional languages typically also have much freer word order than English so that writers are more likely to change the syntactic structure during revision, but moving some words may require adjustments in other parts of the sentence.

Our target group are *experienced writers*, i.e., the functions we are proposing are intended for supporting writers who know how to write and who are used to expressing their writing and revising actions in exact terms. When writers think and talk about texts, they do this on the level of *linguistic units*, not on the level of *characters*, even though they don't necessarily use linguistic *terms*, such as *genus verbi*. So, for example, they may say, "Let's put this sentence into passive voice". Other examples may be "This phrase should be plural", "This sentence is too long, break it up", "Merge these two sentences", or "We should use 'laptop' instead of 'personal computer"'.

3. Improving word processing – related work and state of the art

Even early editors had functions operating on other units than characters and lines, viz. "words" and "sentences" (cf. van Dam and Rice (1971), Callender (1982)). These units were, however, only defined in terms of the writing system, i.e., a "word" was defined as a string bounded by whitespace

[1] See (Mahlow and Piotrowski, 2008, p. 639) for a detailed description.

or punctuation, and a "sentence" was defined as a string terminated by a period followed by two space characters.

There has been research on automatic spelling, grammar, and style checking at universities and research institutions ever since computers started being used for writing natural-language text; Cherry (1981) provides an overview of some early research. With the spread of commercial word processors with graphical user interfaces and integrated checkers most of these research systems disappeared (cf. Vernon (2000, p. 332)). After several years of using commercial software for composing – and for teaching how to compose – the need for more suitable functionalities arose again, along with criticism of the functionality of word processors and checkers for spelling, grammar, and style (see, for example, Piolat (1991), Dale (1997), Vernon (2000), McGee and Ericsson (2002)). But today, more than one decade later, we are still lacking better functions. Popular word processors still operate on non-linguistic entities and checkers for spelling, grammar, or style are still unreliable.

Today, word processors, whether commercial or open-source, whether used for print or for the Web, all have very similar graphical user interfaces. You will find nearly identical buttons and menus for nearly identical sets of functions – setting text in italics, changing the text color, switching from flush left to fully justified alignment, etc. Writers are accustomed to these functions and to this feature set. Sharples and Pemberton argued in 1990 already that writers "can infer the existence of operations and how to perform them from previous experiences" (Sharples and Pemberton, 1990, p. 49). At this time, the use of word processors was not as common as it is today, and today's writers are even more used to core operations as *cut*, *copy*, *paste*, etc. Since users are conditioned on the core operations and their behavior, experimental editors would have to behave in the same way to be accepted.

Research projects that proposed new and improved functionality by designing completely new editors with a completely different look and feel (see, for example, Dale (1990; Williams (1990; Holt et al. (1990; Sharples and Pemberton (1990; Dale (1997)) didn't have much success, and certainly no lasting impact. We thus think writers have to be offered new and – in our opinion – more suitable functions for professional writing, revising and editing as an *extension* to their preferred word processor or editor. In the 1990s, performance issues may also have been a motivation for the creation of new editors, but we now have enough computing power for on-the-fly analyses and operations, and we can thus take a different approach.

4. Approach

Writers are used to the functions that their editor of choice provides. Having the possibility to directly compare the new functions with the previous way of performing an operation should promote the acceptance of the new functions and help with the evaluation. We are therefore implementing new functions in the XEmacs[2] editor, which will thus serve as a test bed for testing new functions with *real* users doing their daily work. They will still be able to use the existing functions and will not have to learn how to use a new editor.

[2] `http://xemacs.org/`

We do not want to create a new editor, and we are not interested in developing another grammar checker (or some other postwriting tool) either. As Sharples (1999, p. 190) points out, advanced writers tend to turn off this type of "assistance", or they try to ignore it (McGee and Ericsson, 2002, p. 462). The problem with checkers is that they are good for ensuring consistency; advanced writers, however, are interested in creating specific effects (whether in terms of vocabulary, syntax, or style) to achieve their specific communicative goals. Advanced writers know what they are doing and they know why they are doing it. We do not want to say that checkers are useless: They are certainly good tools for detecting possible errors and for ensuring consistency. However, for advanced writers, these tasks come typically very late in the writing process (in the postwriting phase) and are just another proofing step.

What we are more interested in is adding *language awareness* (as in editors for programming languages) and creating functions that support writers *during* the revision and editing process by simplifying complex operations. Each of these functions should assist the writer in one specific editing task that is tedious or error-prone to carry out manually. The support must be designed in a way that makes it a better alternative. Thus, it must not interfere with the writing process and it must not make any assumptions about what the writer *might* want to do, or whether this is "correct", but it must perform the task quickly, *reliably* (another problem with checkers is that they are not reliable), and under the control of the writer. Also, as Williams (1990, p. 7) points out, the functions have to be interactive to be useful during the process of composing.

4.1. Types of functions

Generally speaking, language awareness can result in two types of functions:

Informational functions Elements – such as words, phrases, or clauses – can be highlighted (known as *syntax highlighting* in programming editors), or the writer can request information about certain aspects of the text, such as prepositions used, conjuncts, sentences without verbs, or variants of multi-word expressions. The writer has to interpret the results himself and can decide how to make use of them.

Operations The other type are functions that operate on and make changes to textual elements. Linguistic elements can be reordered, modified, or deleted. In order to reduce the cognitive load, it is desirable to reduce the number of actions necessary to reach a specific goal (Allen and Scerbo, 1983). This can be achieved by combining sequences of core operations into higher-level functions closer to the users' goals and their mental model of the task. These functions always behave in the same way and none of the involved core operations will be forgotten or executed twice. The cost of the "brain-to-hand-to-keyboard-to-screen" process is thus reduced.

For both types of functions we can find examples requiring only minimal linguistic knowledge. Functions requiring

more linguistic knowledge differ with respect to the required resources: In some cases, only static resources, e.g., a list of all conjunctions of a language, are necessary. In other cases, morphologic analysis and/or generation may be needed at run time. Sometimes an operation may even need syntactic knowledge about the context. Since sufficient computing power is available today to perform linguistic analyses on the fly, and since linguistic resources, such as morphologic components, are available for different languages at a reasonable cost or even as open source, it is now realistic to employ them in an interactive editing environment.

For example, highlighting conjunctions may help writers in getting an overview of their argumentation structure. Executing the command `show-conjunctions` would give a quick overview of the use of conjunctions. To implement this function, only minimal linguistic resources are required. As conjunctions are typically invariable, there is no need to look for different word forms, and since conjunctions are not linguistically productive (i.e., no new conjunctions are produced using derivation or composition), it is not necessary to consider morphological processes. Thus, only a list of conjunctions is needed.

Detecting sentences without verbs is a more complex example of an informational function. Here, linguistic resources are clearly necessary: First, sentence boundaries must be identified, and then the POS of the word forms of each sentence have to be determined. Thus, this can be considered a more advanced function.

For operations, swapping conjuncts may serve as an example requiring only minimal linguistic resources. This function, as mentioned in section 2., requires certain core operations, depending on the editor. This type of function is aware of (and dependent on) the writing system but needs no further linguistic knowledge, e.g., of the word classes involved – and we do not want to restrict writers in what they can transpose.

At the other end of the spectrum would be a function for replacing words, i.e., all word forms of a word should be replaced with the corresponding word form of another word. However, in inflectional languages, like German, words can have many word forms and each word form can typically express more than one category (see table 1 for the paradigms of two German nouns).

Manually replacing all occurrences of *Zelt* 'tent' with the corresponding word form of *Haus* 'house' is therefore a complex task: First, one has to find all word forms of *Zelt* – with the usual search functions this will require to search for each word form individually. Then, one has to determine the category of a specific occurrence; note that the word form may be ambiguous, and the exact category can only be found by looking at the syntactic context. Finally, one must manually replace the word form of *Zelt* with the corresponding word form of *Haus*. [3]

We are thus proposing a function `query-replace-word`, which would operate as follows: After calling the function, the writer is prompted to enter the word to replace (*from-word*) and its replacement (*to-word*). The function searches for all forms of the paradigm of *from-word*; when a form of *from-word* is found, it is replaced with the corresponding form of *to-word*. It is clear that this task requires morphological analysis and generation. In fact, replacing the word form *Zelte* with the corresponding word form of *Haus* requires even syntactical analysis: *Zelte* can be of the categories DatSg, NomPl, GenPl, AccPl, but *Haus* has different word forms for DatSg on the one hand and for NomPl, GenPl, AccPl on the other hand.

Word	**Forms**	**Categories**
Zelt	Zelt	NomSg,
(*n*, (e)s/e decl.)		DatSg, AccSg
	Zeltes	GenSg
	Zelts	GenSg
	Zelte	DatSg, NomPl,
		GenPl, AccPl
	Zelten	DatPl
Haus	Haus	NomSg,
(*n*, (e)s/er decl.)		DatSg, AccSg
	Hauses	GenSg
	Hause	DatSg
	Häuser	NomPl, GenPl,
		AccPl
	Häusern	DatPl

Table 1: Word forms of *Zelt* and *Haus*.

4.2. Components of a pilot implementation

We have chosen XEmacs as an implementation testbed for the following reasons: It is open-source and new functions can easily be added using Emacs Lisp, either as additional functions or replacing existing functions.

All functions – informational functions as well as operations – will be implemented in a new minor mode called *natlang-mode*. This mode can then be used with various major modes, such as *LaTeX-mode*, *text-mode*, or *message-mode*, i.e., in all modes dealing with natural-language texts. The syntax highlighting facilities can be used for implementing various informational functions.

As usual in XEmacs, functions can either be bound to keystrokes, e.g., `C-c C-r` for `transpose-conjuncts`, or by called by name, e.g., using `M-x transpose-conjuncts`. Functions can also be called by selecting them from a menu. Since these functions will be implemented in a similiar way existing functions are implemented, optional or required parameters can be specified in the usual ways.

We make use of the internal handling of XEmacs for words and sentences: A "word" is a character string bounded by spaces, a "sentence" begins with a capitalized word and ends with a period. To distinguish this period from the ones in abbreviations in an easy way, we require writers to follow sentence-final periods by two spaces, as it is commonly done in English, even though this practice is not normally used in German. Of course, in a more advanced implementation we will use a proper sentence-detection component. By considering spelling German conventions, we can also extract some basic information from the text: Nouns and proper names are always capitalized, while all other word classes

[3] For a discussion of side effects of such functions see (Mahlow and Piotrowski, 2008).

start with a lower case letter except when at the beginning of a sentence.
The morphological component we use for German is DMM (Deutsche Malaga-Morphologie) (Lorenz, 1996).

5. Towards a taxonomy of language-aware functions

In section 4.1. we have presented some first thoughts on linguistic support for revising and editing. However, we still need to get a better idea of what is actually happening when writers are revising and editing. The processes can be described with respect to various dimensions, some of which are:

Destructiveness As described in section 4.1., we distinguish between *informational functions* and *operations*. Informational functions highlight elements to help the writer in finding certain occurences of a linguistic element or phenomenon. They show results of certain analyses, e.g., the number of sentences lacking a finite verb or variants of multi-word expressions. Operations modify linguistic elements such as words, phrases, or clauses. *Destructivness* is a boolean attribute.

Level of language dependence The type of the required linguistic resources determines the level of language dependence of a function.

For functions like `transpose-conjuncts` it is sufficient to use the core operations of a word processor and the properties of the writing system of a certain language. It is not necessary to analyze the words affected by this function, and the definition of a *word* as a string bounded by whitespace or punctuation is completely sufficient for functions that transpose words or move the cursor. Thus, functions using basic concepts and principles of a certain language and core operations will have a low level of language dependence.

Functions like `show-conjunctions` have a higher level of language dependence: Linguistic resources are required, in this case a list of conjunctions. In general, highlighting functions, or functions operating on invariable elements, need similar linguistic resources, i.e., word lists or lexica, but they do not need explicit morphological or syntactical rules. We call these linguistic resources *static*.

The highest level of language dependence includes functions like `query-replace-word`. Here, morphological analysis and generation and, in some cases, even syntactic analysis are involved. The linguistic resources have to be in a form suitable for run-time execution. We call these linguistic resources *dynamic*.

The *level of language dependence* is not an absolute value. It can only be determined by comparison with the requirements of other functions.

Complexity We measure *complexity* in terms of core operations required. The value for *complexity* can be specified as an absolute number, or it can be specified on a scale between *high* and *low* by comparing it with the corresponding values of other functions.

We have to take into account that a certain function can be split into *core operations*, i.e., atomic editor operations, or into *involved operations*. If we combine some of our additional functions, each consisting of a certain sequence of core operations, into even more complex functions, the constituent functions will be called *involved operations*.

Specificity The kind of arguments a function takes determines the *specificity* of the function. Some functions take a *concrete element*, such as a word like "house", while others operate on *classes* like conjunctions, finite verbs in preterite tense, or relative clauses. The value for *specificity* is an element from a set of argument types.

Area of operation We use this term to describe whether a function operates on one specific occurrence of a linguistic element or whether it operates on all occurrences of an element. For example, a function such as `transpose-conjuncts` only operates at the current cursor position, whereas a function such as `query-replace-word` is designed to operate on all occurrences of the specified *from-word* (even if the user chooses to interrupt the function at some point). The *area of operation* is either *local* or *global*.

Side effects When executing an operation it can happen that the result is not grammatical. This is clearly a unintended *side effect*. Such an operation will thus require further editing operations on other elements to restore grammaticality. *Side effects* can be of different types, such as agreement errors or dangling anaphoric references. *Side effects* can also occur at different distances from the original operation – they may be restricted to the same phrase (e.g., adjective-noun agreement), or they may occur further away, even in other sentences.

We can distinguish the values *has side effects* and *has no side effects*. For operations with side effects we can determine the type of side effects as one element from a set of types.

To be able to determine these dimensions it is necessary to analyze and to classify user actions, e.g., from keystroke recordings (cf. Good (1985), Flinn (1987), Perrin (2006)), and to consider linguistic rules and engineering issues.
The classification can serve as a help in making decisions, and it can then be used to write specifications and serve as a guideline for the actual implementation.

6. Conclusion and further work

We think that language-aware editing functions can relieve writers from many low-level operations which distract from the actual revision and editing process. Our approach concentrates on support *during* the writing process, enabling the writer to interact with the word processor.
We showed some examples for such functions, concentrating on German. These functions can be seen as add-ons to existing word processors, allowing writers to use the functionality they are accustomed to and benefit from the new ones. The required resources are relativly small but

can have a considerable impact on the writing process. We then outlined a number of aspects which have to be taken into consideration, and which should eventually result in a taxonomy of revising and editing operations.

Currently we are selecting the operations to be implemented according to the principles described in this paper. We will then evaluate them with experienced writers.

Once we have an implementation serving as a proof of concept, we will then be able to consider additional aspects: What is the best way to make writers learn new functions? What is the best way to call these functions: using keystrokes or using pull-down menus? Which linguistic *terms* do writers really use when talking about linguistic *units*, and which terms should be used in editor functions?

Acknowledgements We thank Yves Forkl and Rolf Schwitter for fruitful discussions.

7. References

Robert B. Allen and M. W. Scerbo. 1983. Details of command-language keystrokes. *ACM Trans. Inf. Syst.*, 1(2):159–178, April.

E. D. Callender. 1982. An evaluation of the AUGMENT system. In *SIGDOC '82: Proceedings of the 1st annual international conference on Systems documentation*, pages 29–35, New York, NY. ACM Press.

Lorinda Cherry. 1981. Computer aids for writers. *ACM SIGOA Newsletter*, 2(1-2):61–67.

Robert Dale. 1990. A rule-based approach to computer-assisted copy-editing. *Computer Assisted Language Learning*, 2(1):59–67.

Robert Dale. 1997. Computer assistance in text creation and editing. In Giovanni B. Varile, Antonio Zamponelli, Ronald Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue, editors, *Survey of the State of the Art in Human Language Technology*, pages 235–237. Cambridge University Press, New York, NY.

Jane Z. Flinn. 1987. Case studies of revision aided by keystroke recording and replaying software. *Computers and Composition*, 5(1):31–44, November.

Michael Good. 1985. The use of logging data in the design of a new text editor. *SIGCHI Bull.*, 16(4):93–97, April.

Thomas Haigh. 2006. Remembering the office of the future: The origins of word processing and office automation. *Annals of the History of Computing, IEEE*, 28(4):6–31.

Patrik O. Holt, Dag C. Hegg, and Terje Johnsen. 1990. Engineering written style. *Computer Assisted Language Learning*, 2(1):27–35.

Oliver Lorenz. 1996. Automatische Wortformerkennung für das Deutsche im Rahmen von MALAGA. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg.

Cerstin Mahlow and Michael Piotrowski. 2008. Linguistic support for revising and editing. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing: 9th International Conference, CICLing 2008, Haifa, Israel, February 17–23, 2008. Proceedings*, pages 631–642, Heidelberg. Springer.

Deborah McCutchen. 1996. A capacity theory of writing: Working memory in composition. *Educational Psychology Review*, 8(3):299–325.

Tim McGee and Patricia Ericsson. 2002. The politics of the program: MS Word as the invisible grammarian. *Computers and Composition*, 19(4):453–470, December.

R. L. Oakman. 1994. The evolution of intelligent writing assistants: trends and future prospects. In *Tools with Artificial Intelligence, 1994. Proceedings., Sixth International Conference on*, pages 233–234.

Daniel Perrin. 2006. Schreibforschung im Kursalltag: Was die Progressionsanalyse praktisch nützt. In Otto Kruse, Katja Berger, and Marianne Ulmi, editors, *Prozessorientierte Schreibdidaktik: Schreibtraining für Schule, Studium und Beruf*, pages 279–294. Haupt Verlag, Bern, Stuttgart, Wien.

Annie Piolat. 1991. Effects of word processing on text revision. *Language and Education*, 5(4):255–272.

Mike Sharples and Lyn Pemberton. 1990. Starting from the writer: Guidelines for the design of user-centred document processors. *Computer Assisted Language Learning*, 2(1):37–57.

Mike Sharples. 1999. *How We Write: Writing As Creative Design*. Routledge, June.

Lee R. Taylor. 1987. Software views: A fistful of word-processing programs. *Computers and Composition*, 5(1):79–90.

Andries van Dam and David E. Rice. 1971. On-line text editing: A survey. *ACM Comput. Surv.*, 3(3):93–114, September.

Alex Vernon. 2000. Computerized grammar checkers 2000: capabilities, limitations, and pedagogical possibilities. *Computers and Composition*, 17(3):329–349, December.

Noel Williams. 1990. Writers' problems and computer solutions. *Computer Assisted Language Learning*, 2(1):5–25.